

AD-A087 355

KRANNERT GRADUATE SCHOOL OF MANAGEMENT LAFAYETTE IN  
FUTURE DIRECTIONS FOR DECISION SUPPORT.(U)  
JUN 80 R H BONCZEK, C W HOLSAPPLE

F/6 5/8

DAA629-79-C-0154

UNCLASSIFIED

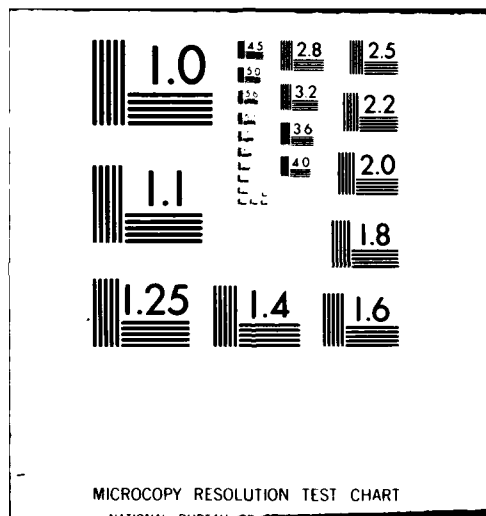
ARO-16231.2-EL

NL

Fig 1  
AD-A087 355




END  
DATE  
FILMED  
9-80  
DTIC



ADA 087355

DDC FILE COPY

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS  
BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A087355	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) FUTURE DIRECTIONS FOR DECISION SUPPORT		5. TYPE OF REPORT & PERIOD COVERED Technical Rept.
6. AUTHOR(s) Robert H./Bonczek Clyde W./Holsapple Andrew B./Whinston		7. CONTRACT OR GRANT NUMBER(s) Contract No. DA79C0154 new DAG-29-79-C-0154
8. PERFORMING ORGANIZATION NAME AND ADDRESS Purdue University Krannert Graduate School of Management West Lafayette, IN 47907		9. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 12 35
10. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office Post Office Box 12211 Research Triangle Park, NC 27709		11. REPORT DATE June 1980
12. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 15 DAG-29-79-C-0154		13. NUMBER OF PAGES 34
		14. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A formal, generic description of decision support is introduced. This description views a decision support system as having three principal components: a language system, a knowledge system and a problem processing system. Several systems are described which fit the generic DSS idea, but which are (for the most part) not the customary kinds of systems encountered in business applications. Nevertheless, the concepts and techniques employed in these systems can make important contributions to the emergence of more powerful business-oriented decision support systems.

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE  
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

80 8 1 058

DTIC  
ELECTE  
AUG 1 1980

A

FUTURE DIRECTIONS FOR DECISION SUPPORT

Technical Report

by

Robert H. Bonczek

Clyde W. Holsapple

Andrew B. Whinston

U.S. Army Research Office

Contract No. DA79C0154

Purdue University

Approved for Public Release

Distribution Unlimited:

Dist		
A		

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

## **FUTURE DIRECTIONS FOR DECISION SUPPORT**


**Robert H. Bonczek**  
Assistant Professor of Management  
Krannert Graduate School of Management  
Purdue University

**Clyde W. Holsapple**  
Assistant Professor of Business Administration  
University of Illinois


**Andrew B. Whinston**  
Professor of Management and Computer Science  
Krannert Graduate School of Management  
Purdue University

**Research Supported in Part by Army Grant Number DA79C0154**


**The authors are indebted to the referees for comments on  
our earlier draft.**



## Abstract



A formal, generic description of decision support systems is introduced. This description views a decision support system as having three principal components: a language system, a knowledge system and a problem processing system. Several systems are described which fit the generic DSS idea, but which are (for the most part) not the customary kinds of systems encountered in business applications. Nevertheless, the concepts and techniques employed in these systems can make important contributions to the emergence of more powerful business-oriented decision support systems.



## 1.0 Introduction

This presentation commences with a formal, generic description of decision support systems (DSS) which will guide the remainder of our study. This description stems from a decision making framework [2] and from the classification scheme introduced in a companion paper [5]. The generic description views a DSS as having three principal components: a language system (LS), a knowledge system (KS), and a problem processing system (PPS). We proceed to examine several computer-based systems which adhere to the generic DSS description. However, most of these systems are not business oriented and some clearly do not lie within the most prevalent category. These various systems are addressed from the angle of identifying techniques and concepts which can be used to enhance present-day, business-oriented decision support systems like those reviewed in [5].

### 1.1 Language System

An earlier paper [ 5] surveyed systems which have models incorporated into them. Even though the descriptions of such systems in the literature typically pay but scant attention to the languages used to direct retrieval and computation, such languages were recognized in the Xerox case [16]. This served as the basis for a classification scheme which stressed the distinction between languages for retrieval and languages for directing computation. It was emphasized that languages for retrieval could be utilized by either the human user or a model. Languages for directing computation were presented from the standpoint of being used by the human elements of a decision making system. However, it is certainly imaginable that one mechanical DSS could direct another mechanical DSS; for the present we shall ignore this possibility.

We shall refer to a language system (LS) as the sum total of all linguistic facilities made available to the decision maker by a decision support system (see Figure 1). A language system may encompass either or both retrieval languages and computational languages. A language system is not concerned with the interfacing of models and data. As indicated in [5], a user's interface language could be so designed that the user is unaware of whether he or she is directing a retrieval or a modeling process (e.g., categories G, H, I, or L in [5]).

A language system is characterized by the syntax which it furnishes to the decision maker, by the statements, commands, or expressions which it allows the user to make. Thus a language system is a vehicle which allows the decision maker to express himself, but at the same time it limits the permissible expressions. We say that an LS is a vehicle in that it is a mode of conveyance, not of material, but of information.



## 1.2 Knowledge System

Unless it contains some knowledge about the decision maker's problem domain, a decision support system is likely to be of little practical value. Indeed a good deal of the power of a decision support system derives from its knowledgeability about a problem domain (e.g., banking). This knowledge typically includes large volumes of facts which the decision maker has neither the time, nor the inclination, nor the opportunity to absorb into his own memory. But certain subsets of this volume of facts are important to a reasonable or good decision in the face of a particular problem from the problem domain.

We shall refer to a decision support system's body of knowledge about a problem domain as its knowledge system (KS) (see Figure 2). In the systems described in [ 5 ], the KS consisted of what were termed data files or data bases. In the original descriptions of such systems in the literature the precise nature of their knowledge systems typically is left unexplained. That they possess some sort of KS is unmistakable and sometimes the kinds of data held in the KS are detailed. But the crucial issue of how that data is organized is left unaddressed, perhaps due to space limitations.

Knowledge represented in a KS must be retained in an organized, systematic manner. There are a variety of knowledge representation methods [ 3 ], [21]. The knowledge representation method utilized by a particular knowledge system may be thought of as a set of rules according to which knowledge is expressed for purposes of retention within the decision support system.

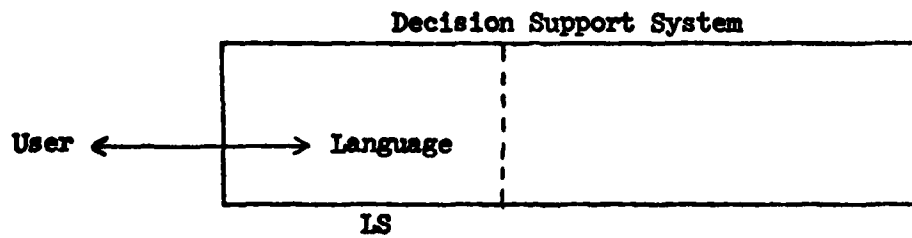


Figure 1. Language System (LS)

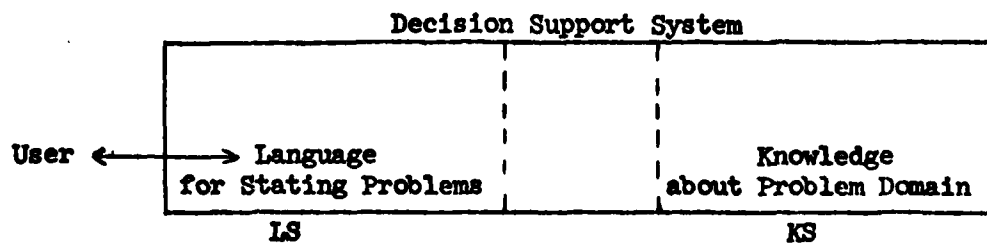
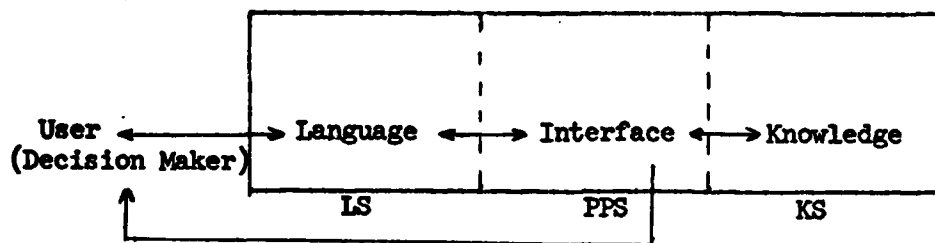


Figure 2. Language System + Knowledge System (KS)

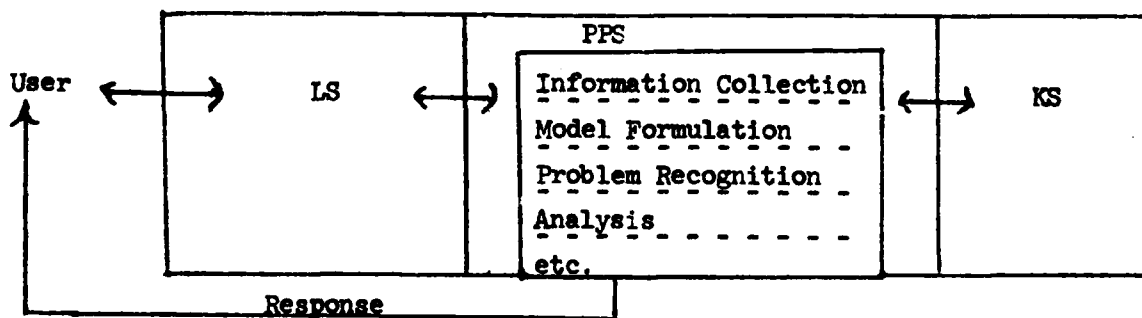
### 1.3 Problem Processing System

The main function of a decision support system is to take strings of symbols organized according to LS syntax (problems) and to take strings of symbols organized according to KS representation rules (problem domain knowledge) and to produce information that supports (enhances or makes possible) a decision process. To do so there must be a mediating mechanism between expressions of knowledge in the KS, and expressions of problems in the LS. We refer to this mediating mechanism as the problem processor or the problem processing system (PPS).

Figure 3a provides a 'bird's eye view' of the generic description of decision support systems. It is the PPS which has one or more of the seven abilities [2] required for decision making. A more detailed representation appears in Figure 3b, where some of the PPS's potential abilities are specified.



a. Generic DSS: Overview



b. Generic DSS: Detail

Figure 3. Generic DSS

#### 1.4 Decision Support Systems: Human and Computer

Referring to Figure 3a, a close analogy may be drawn between a computer-based decision support system and a human-based decision support system. The former is implemented with a computer, whereas the latter is implemented with staff personnel or human consultants. In each instance, however, the decision support system must contain an LS, KS, and PPS.

In the computer-based case the PPS is computer software. The PPS in the human-based case consists of the mental skills of staff ('human software'). If decision support is to be provided, the software must be able to understand the decision maker's requests as stated in some language and it must be able to extract pertinent information from some available pool of knowledge about the problem domain. The function of the software (be it human or computer) is to interface these two so as to produce an answer to the problem posed by a decision maker.

In some systems, interfacing software may be simple, whereas it may be quite complex in others. On another dimension some software may be able to accept statements of problems in many languages; other PPS may be more specialized, accepting statements in one language only. One yet another dimension, some software interfaces are specially constructed to access knowledge of one problem domain only. Other PPS have knowledge of many problem domains (i.e., they can interface with many knowledge systems). On still another dimension, some problem processing systems may have models embedded within them (e.g., Potlatch, Banking System in [5]). Other PPS may be separated from modules which are stored in the KS, but the PPS 'knows' how to use these to formulate models. This latter kind of PPS is along the lines of Simon's "general strategies" discussed in [18].

In both computer-based and human-based decision support systems there is wide variation in the nature of the language used to direct the decision support system. In some decision support systems of each type there is the possibility of the PPS misunderstanding the intention which the decision maker is endeavoring to convey through the available language system. There are decision support systems of each genre which have interactive language systems. That is, the decision support system's PPS is permitted to query the decision maker for further information in order to clarify what it recognizes as ambiguity in the decision maker's initial statements. Thus in a decision support system of this type communication via the LS is not a one-way avenue, rather the LS is a two-way avenue through which the decision maker and the PPS can interact.

## 2.0 The Shape of Systems to Come

Although most were not developed with business applications in mind, the systems described in the remainder of this paper do fit the generic pattern of a DSS displayed in Figure 3a. (As an exercise the reader can verify that systems reviewed in [5] also fit the pattern.) The systems presented here have certain novel features which are not encountered in typical business-oriented systems. The descriptions given here are intended to provide overviews.

The first system discussed was specifically devised to aid managers in the model formulation activity. The next three systems have been drawn from the artificial intelligence literature. Some fairly general ideas on problem solving have been developed in the field of artificial intelligence. Since the ideas of this discipline are oriented towards solving semistructured or unstructured problems it is useful to explore the basic concepts, with an eye to their possible applications in the decision support field.

The artificial intelligence systems are often characterized as being theorem proving systems. Use of the term "theorem-proving" does not necessarily connote proving theorems in mathematics. It is used in a broad sense which may be interpreted variously, not only as a theorem to be proven, but also as a goal to be achieved, a problem to be solved, or a query to be answered.

Finally there is a description of a decision support system that is more general than the systems reviewed in [5]. That is, the system can be tailored to treat different problem domains such as banking, project management, corporate planning, etc.

## 2.1 The IBM Business Definition Language

Here we examine a system for supporting business decisions which is approached from the angle of a language [10]. The language has been designed to provide business people with a means for stating their problems and stating how they are to be solved. This Business Definition Language (BDL) deals with four kinds of elements: documents, steps, paths, and files.

A document is a collection of data values organized according to some form. An example of a form is an invoice form (with no data values filled in). A particular group of data values properly written into the invoice form is an invoice document.

A step is a procedure which converts some input documents into some output documents. An irreducible step may be viewed as an operator that is directly available to the user of BDL. Paths connect steps by indicating that output documents of one step are input documents to another step.

The BDL user commences by stating the types of output documents that must be produced from a particular collection of input documents. The user must eventually specify an algorithm (program) for accomplishing this, in terms of paths connecting irreducible steps. The resultant program is a composite step in that it is composed of more primitive steps. See [10] for a description of how the program construction process is interactively guided by a series of reductions which break composite steps into more primitive steps, until all primitive steps are irreducible.

Notice that the BDL approach to directing computations differs from that of systems reviewed in [5]. In those systems the user either called for a model (Potlatch) or specified a sequence of models (Xerox). The BDL



language systems is more procedural, but more flexible than the language system of Potlatch or Xerox. It falls nearer to the top of the classification matrix in [ 5] than do these others. Like these other systems the BDL PPS contains models (i.e., irreducible steps or modules). The BDL knowledge system consists of files.

Input documents to a BDL program may be directly provided by the BDL user or they may be extracted from files. A file is composed of documents of a single form which are more or less permanently maintained.

BDL has been described [10] as having three major "components." These are the Form Definition Component (FDC), the Document Flow Component (DFC) and the Document Transformation Component (DTC). The first (FDC) specifies the forms of documents which can be used in the BDL program. The DFC is used to specify the BDL program in terms of paths and steps. Irreducible steps are programmed using a language contained in the DTC component.

BDL is being developed in order to reduce the cost of labor-intensive application software. The user of BDL is interactively guided through a process of program (model) construction, beginning with a composite step and eventuating in a sequence of irreducible steps. (The issues of branching and iteration in a resultant program are not dealt with in [10]. However, branching and iteration do occur within irreducible steps. It is important to observe that the irreducible steps are possibly complex programs.) The thrust of the BDL development is directed towards easing the tasks of writing programs in a business problem domain.

## 2.2 PLANNER

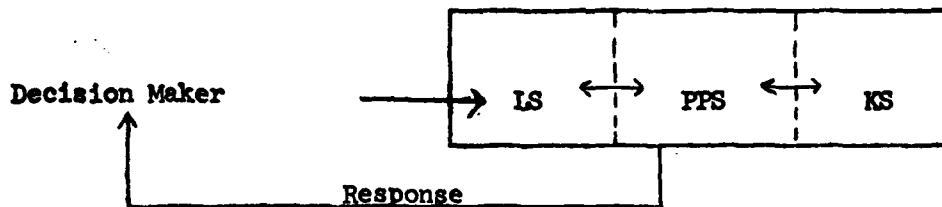
One of the most prominent theorem-proving systems is PLANNER, [20] which is under continuing development. In general, a theorem proving system consists of 1) some language for stating the theorem to be proven (LS), 2) some collection of axioms, assertions, or formulas embodying knowledge about a problem domain which the system is concerned with (KS), and 3) a theorem prover (PPS) which utilizes principles of logic to endeavor to deduce a stated theorem from problem domain knowledge.

A major point of variation among theorem proving systems is the method for handling knowledge about how to perform the deduction involved in a particular proof, i.e., knowledge beyond those general principles of logic which are applicable to any proof. This 'reasoning' knowledge is not to be confused with the problem domain knowledge mentioned in the preceding paragraph, since the former is knowledge about how to use the latter.

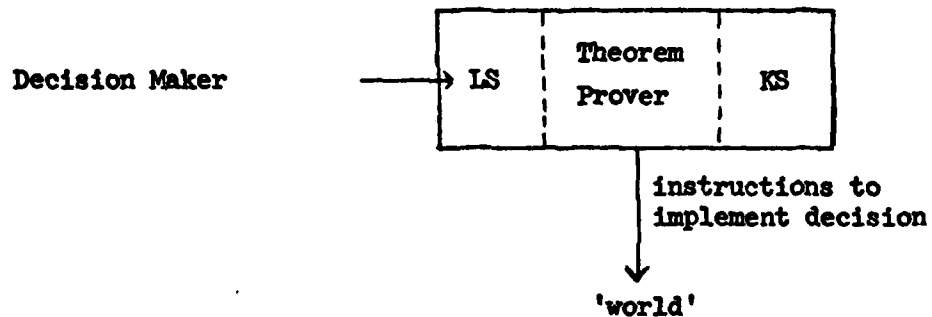
As the problem domain knowledge becomes large, the exclusive use of general deduction principles to prove a theorem tends to bog down, due to blind examination of the possibly large number of plausible proof procedures for a given theorem; many false starts are involved. Indeed this was the source of some discouragement with the practicality of the theorem proving field circa 1970 [19]. One remedy is to incorporate into a theorem proving system, more specialized reasoning knowledge, i.e., special deductive principles which are applicable to a certain problem domain, to a certain type of theorem, or even to particular theorems themselves.

This specialized reasoning knowledge could be embedded directly into the theorem prover (PPS) as a set of heuristics. Another conceivable approach would involve the storage of this knowledge in the form of rules (in the KS) which would be accessible to the theorem prover for use in governing the deduction process. The PLANNER approach is to have the system's user state this specialized reasoning knowledge (in the LS) at the same time as (actually, as a part of) the theorem to be proven. The philosophy underlying the PLANNER approach is that at the time of stating a theorem to be proven, the user also has some notion about how to carry out the deduction. Thus in the PLANNER language a theorem is really a program for guiding the deduction process, giving the user control over how to prove the theorem (i.e., how to use the problem domain knowledge vis a vis the theorem).

The primary problem domain in which PLANNER has been applied consists of a world of blocks of various colors, sizes, shapes, and locations which is inhabited by a robot that can act upon those blocks. Questions or 'theorems' posed by a user can request information about the blocks world; this is obtained either directly from the problem domain knowledge or it is deduced (inferred) from that knowledge (see Category J in [ 5]). A second type of question, problem, or 'theorem' can request a change in the state of the blocks world. This entails a deduction of whether the requested change is possible, given the current state of the blocks world and given the actions which the robot can carry out; these two "givens" are included in the system's problem domain knowledge. A proof of this second type of theorem results in the execution of robot actions.



a. System Oriented Towards Analysis to Support a Decision



b. System Oriented Towards Implementation of a Decision

Figure 4. Decision Support Versus Implementation

Although this PLANNER application is not oriented towards the direction or execution of computations of interest to managers, the execution of robot actions in response to a user's problem is along a similar vein. That is, each involves a plan, but the plans are of different natures. One is a plan of computational analysis (i.e., a model) which, when executed, results in some data (refer to abilities III and V in [2]). The other is a plan of implementation which, when carried out results in some change in a 'world' which can be manipulated by the theorem proving system (refer to abilities III and VII in [2]). This contrast is depicted in Figure 4. (Recall that PLANNER can also supply data to the user via inferential retrieval, as discussed earlier.) As a final comment, notice that the architecture in Figure 4 could be altered, to allow the decision maker to either approve or disapprove of the implementation plan devised by the theorem prover, prior to its execution.

### 2.3 The GPS Approach and STRIPS

A fairly general and abstract approach to problem solving has been developed by Simon [17]. This approach appears in the guise of the General Problem Solver (GPS). The GPS concepts can be easily outlined, although applying them to give an operational system can be quite complicated.

GPS is concerned with three basic elements: states, operators and goals. A state is simply a situation. Operators permit us to move from one state to another state, to transform one situation into another situation. A goal is a desired state. A problem is described in terms of an existing, initial state, and a goal.

The GPS concept of problem solving is characterized as selecting a series of operators that will successively move the problem from its initial state, through intermediate states, to the goal state. Given any state, it is assumed that a difference can be determined between that state and the goal state. Operators are selected to reduce the difference between the current state and the goal state. However, not any operator can be selected, since for a given state the set of applicable operators is a subset of the set of all operators. If, for example, the current state has a salesman in Chicago, then the operator to fly from New York to Albany is not applicable.

One interesting aspect of the GPS approach is that it can be used recursively. To explain this suppose that we are in a particular state (e.g., in Chicago) and when calculating the difference with the goal state (e.g., in Albany) we notice that a particular operator (e.g., fly

from New York to Albany) if applied, would substantially reduce or eliminate the difference. However, to use this particular operator we need to be in a certain state (e.g., in New York) which we presently are not in. Thus we could set up a new goal (e.g., in New York) which would be to achieve a particular state which would allow the application of the selected operator (e.g., fly from New York to Albany). For the solution of this new problem (e.g., initial state: in Chicago, goal state: in New York) we can reapply the GPS approach. If a solution were found (e.g., fly from Chicago to New York) then we can return to solving our original problem. During the course of solving our new problem we may have again had recourse to using GPS in such a recursive fashion (e.g., fly from Chicago to Cleveland and then fly from Cleveland to New York).

A description of an implementation of the GPS approach and its use in solving a number of problems ranging from the cannibal-missionary problem to symbolic integration appears in [7]. For comments on the relationship of the GPS approach to human problem solving see [13]. Notice that the GPS approach involves several of the decision making abilities presented in [2]. These include the collection of information about states and the formulation of models (sequences of operators) as a result of recursively recognizing problems.

The emphasis in GPS is therefore upon the use of the means-end (operators-goal) approach and recursive problem solving in order to automatically determine a method of solution. The principal outputs of systems like those reviewed in [5] are reports such as income statements or sales forecasts. The principal outputs of systems developed in the spirit of GPS are solution methods in terms of operator sequences.

from New York to Albany) if applied, would substantially reduce or eliminate the difference. However, to use this particular operator we need to be in a certain state (e.g., in New York) which we presently are not in. Thus we could set up a new goal (e.g., in New York) which would be to achieve a particular state which would allow the application of the selected operator (e.g., fly from New York to Albany). For the solution of this new problem (e.g., initial state: in Chicago, goal state: in New York) we can reapply the GPS approach. If a solution were found (e.g., fly from Chicago to New York) then we can return to solving our original problem. During the course of solving our new problem we may have again had recourse to using GPS in such a recursive fashion (e.g., fly from Chicago to Cleveland and then fly from Cleveland to New York).

A description of an implementation of the GPS approach and its use in solving a number of problems ranging from the cannibal-missionary problem to symbolic integration appears in [7]. For comments on the relationship of the GPS approach to human problem solving see [13]. Notice that the GPS approach involves several of the decision making abilities presented in [2]. These include the collection of information about states and the formulation of models (sequences of operators) as a result of recursively recognizing problems.

The emphasis in GPS is therefore upon the use of the means-end (operators-goal) approach and recursive problem solving in order to automatically determine a method of solution. The principal outputs of systems like those reviewed in [5] are reports such as income statements or sales forecasts. The principal outputs of systems developed in the spirit of GPS are solution methods in terms of operator sequences.



Now suppose that we use the term "state" in the sense of a state of information (e.g., economic assumptions) and we take operators to be modules (e.g., regression, simulations, etc.). Note that a module enables us to move from one information state (input) to another state (output). If our goal state is one of having a sales forecast, then a GPS approach would yield a sequence of operators (a model) that enables us to arrive at that goal. Incorporation of the GPS approach into a DSS, would therefore go a step beyond the BDL system in which the user sequences the operators. Techniques for incorporating a GPS-like approach into a system which also executes the sequence of operators are currently being investigated [4].

One implementation of a problem solving system based on the GPS ideas was developed at Stanford Research Institute [8] and named STRIPS (Stanford Research Institute Problem Solver). Since the implementation was based on using the predicate calculus and theorem proving, we do not examine STRIPS details here. However, we can sketch out some of its main ideas in the context of our GPS discussion.

The STRIPS problem domain consists of a robot which must rearrange objects, navigate through rooms, etc. A state for the robot problem solver consists of a large number of facts about positions of the robot and objects, about the characteristics of the objects, and about walls and openings in the rooms which the robot can inhabit. Each STRIPS operator has a corresponding "action routine" which when executed, causes the robot to take a certain action. Problem solving consists of finding a sequence of operators which will transform an initial state into a goal state.

In STRIPS, associated with any operator, there are three lists of information. A prerequisite list specifies what must be true about a state before the operator can be applied. The delete and add lists specify what changes to make in the current state if the operator is applied, i.e., what facts should be added and which should be deleted.

The language system (LS) in STRIPS allows for the specification of goal states. Although the language used is predicate calculus, its statements can be paraphrased into English. An example is "Get boxes B and C to location A." The STRIPS knowledge system (KS) consists of a description of the current state of the robot world. These facts are also represented in terms of the predicate calculus.

The problem processing system (PPS) in STRIPS finds (if possible) a sequence of operators which permit the goal to be met. The PPS uses theorem proving techniques to determine whether a goal is met by the current state. If it is, then no operator sequence needs to be determined. If it is not, then the GPS notion of 'differences' is used, along with the previously mentioned operator lists, to select an operator that would give a new current state which is 'closer' to the goal. The PPS then uses theorem proving techniques again to determine whether the goal is met by the new current state. This process continues until a sequence of operators has been found which gives a state that can be proven to meet the goal.

Heuristic intricacies involved in the selection of operators appear in [8]. This can be contrasted with PLANNER, wherein determination of a sequence of robot actions is guided by the user via the PLANNER LS. It can be further contrasted with the BDL approach of the user being guided by the system in eventually specifying a sequence of irreducible steps.

## 2.4 MYCIN

A very specific and apparently successful consultation system called MYCIN [6] is intended to help physicians diagnose and decide upon treatments for infectious diseases. Given an initial set of symptoms and basic information on the patient, the system acts as a consultant by requesting what it considers to be pertinent data about the patient such as results of certain laboratory tests. The acquisition of each new bit of data about a patient serves to influence the next piece of data which the 'consultant' requests from the physician. In this way a selective "data base" is gradually built.

The patient-specific "data base", along with some knowledge about infections, allows the MYCIN system to suggest certain possible diagnoses and therapies, each with a likelihood factor. The physician is aided in the search for a diagnosis by the system's suggestions and by the ability on the physician's part to request reasons for the proposed diagnosis. User interaction with MYCIN is via an English-like interface which significantly enhances its usability.

Although the features of MYCIN differ somewhat from those of the previously discussed systems, it too fits the generic description of a DSS. A consultation session is initiated by the user, but it is perpetuated and guided by MYCIN. Thus MYCIN's information collection from a user via the LS is instigated by the MYCIN PPS (see [6] for examples of this). In contrast to the three previously described systems, the MYCIN LS is quite English-like in appearance. However, the user is largely restricted to asking only one kind of question, namely "What is the diagnosis and therapy?"

The PPS endeavors to condition this query by building a patient-specific "data base" as described earlier, so that the query begins to look like "What is the diagnosis and therapy for patient A having history B, symptoms C, D and E, lab test results F and G, etc?" This interactive conditioning process may be characterized as the MYCIN PPS's problem recognition ability [2].

The exercise of this ability depends upon facts about infections which are maintained in MYCIN's knowledge system (KS). During the interactive conditioning of a user's query, the next bit of data to be collected from a user (to further define or condition the problem) depends upon information already collected from the user (IS) and from the KS. The patient-specific "data base" which results from interactive problem recognition may be considered either as information internally held within the PPS or as a short-term part of the KS that is filled in as data is collected from a user.

The KS holds long term (yet modifiable) knowledge concerning infections which exists across many consultation sessions. The patient-specific information, which the MYCIN authors call a "data base" exists only for the consultation session involving that specific patient. A new patient demands a new patient-specific "data base". In this sense, a MYCIN "data base" is analogous to the internal representation of a particular query in systems such as those reviewed in [5]. Both are comparatively ephemeral.

It is instructive to contrast the MYCIN handling of patient information against another conceivable approach which would store all patient information in the KS on a long term basis. Where the potential patient population is large, the MYCIN approach is clearly superior, since there

would be too much data that is used too infrequently (and which is too volatile) to permit feasible long term storage. The MYCIN approach is entirely consistent with the way in which the decision support systems of [5] handle very short term information. Examples of such short term information include the assumptions under which some sales forecasting model should be executed. We may want to execute the model assuming a 5% economic growth factor and a moment later we may want to try a 7% factor. Just as it is not practical to store all patient data on a long term basis, it is not practical to store all potentially interesting growth factors on a long term basis. On the contrary they are supplied by the user as needed through the system's IS.

Even though MYCIN cannot quite be characterized as permitting computational management (recall the matrix of [5]), its approach to decision support has much to offer in terms of ideas for DSS development in business settings. MYCIN lies in Category L and its inferential retrieval is based upon Post's production system [14]. The notion of diagnosis and treatment of human patients can be extended to business firms. This expanded perspective would see corporate planners as 'physicians' endeavoring to diagnose and recommend therapies for a firm's 'ills'. An interesting issue for research is the extent to which the MYCIN principles can be applied to situations where the patient is a business firm. This would necessitate a KS that deals not with medical knowledge, but with knowledge about the anatomy, testing procedures, afflictions, and treatments for a business firm.

## 2.5 GPLAN

The Generalized Planning System (GPLAN), developed at Purdue University [11], is more general than the systems reviewed in [ 5] in terms of the problem domains it can handle. GPLAN can be tailored to support decision makers in any of a variety of areas (e.g., inventory management [1], water quality planning [12]). The intent was not only one of generality, but also involved an easy-to-use user language, a flexible knowledge system and a problem processor which could automatically interface a model with some data in the KS to perform a desired analysis.

The language system (LS) was designed for use by persons who are not programmers. It is English-like in appearance and with it a user can direct both data retrieval and computation non-procedurally. When used for data retrieval GPLAN falls into Category L. When used for directing computations the system, as currently implemented, falls into Category E. That is, models are invoked by name and they obtain needed data from the KS by invoking special report generators.

Part of the system's generality derives from the fact that the LS syntax remains the same, regardless of the problem domain. The syntax is:

< COMMAND > < FIND CLAUSE > < CONDITIONAL CLAUSE >

The vocabulary does change from one problem domain to another (see [1] and [12] for example). But since the GPLAN problem processor handles user statements using syntax directed compilation techniques, vocabulary changes do not drastically affect the PPS.

On the other hand changes in the models to be used do affect the PPS in several ways. First of all the models are treated as part of the PPS. So a change in problem domain from inventory management to water quality

planning would cause the deletion of inventory management algorithms and the inclusion of water quality simulation models. There are other models which remain in the PPS regardless of the problem domain. These are statistical models, of the sort found in SPSS, which have wide applicability.

The incorporation of a new model into the system affects the LS vocabulary by adding a new command (e.g., RUN WATER SIMULATION) and it necessitates other changes in the PPS such as the possible inclusion of additional report generators and checks to assure that the user's CONDITIONAL CLAUSE fully specifies the assumptions (if any) under which the model is to be executed. Thus the modeling aspect of GPLAN is its least general ability. This can be overcome as described in [ 4].

A final aspect of the system which is quite general is its ability to gather information from the KS (which is a type of data base). The PPS is unaffected by the problem domain or the way in which problem domain knowledge is organized in the KS, having the ability to produce practically any subset of the data contained in the KS. In [ 3 ] it is shown how to extend the power of data bases for representing knowledge.

### 3.0 Rationale for the Study of a Generalized Problem Processor

The systems surveyed in this and its companion paper [5] serve to portray many of the major features and issues pertaining to decision support systems. The concepts and techniques on which these systems are founded furnish several of the tools needed for the development of a general PPS, that is a single, invariant mechanism, whose code does not change with the problem domains which it supports. This implies that such a processor, while possessing various abilities involved in decision making, must be separated from anything which is domain-specific. The design of such a processor is the subject of current research.

It is appropriate at this juncture to pause for a consideration of the advantages of such a generalized problem processor vis-a'-vis domain-specific processors. Perhaps the most significant attributes of the generalized approach is the conceptual framework which it affords. This is particularly important from the pedagogical standpoint. Given a knowledge of the general system, one is in a much better position to comprehend and derive specialized systems than would be the case in the absence of such knowledge. Observe that possession of this knowledge furnishes a basis for systematic comparison, contrast and discourse regarding special cases.

Of course we do not mean to suggest that specialists be deterred from their lines of investigation and development. Indeed, the specialist is quick to point out that with respect to execution and storage, a specialized system is almost invariably more efficient than a general system. It must be remembered, however, that the price for greater operational efficiency is the forfeit of some degree of flexibility. Where efficiency is paramount it may be simpler or less costly (more efficient?) to constrain an existing general system to conform with some standard of



operational efficiency than to devise a special system from scratch. In this connection, it is important to keep in mind the tradeoff between development costs (which continually rise due to the high labor component) and execution plus storage costs (which continue their dramatic decrease).

As a topic for research (and as a managerial strategy) there is a good deal to be said for concentrating on how to improve responses to needs over time, under changing conditions. In the long run this may very well be more fruitful than devotion to augmenting one's capabilities for finding solutions that are optimal at a given moment of time. This would imply a need for flexible systems that can easily adapt to what has been learned and to current needs. One is faced with two distinct courses of action: 1) investment of resources in a system that is optimal until conditions change, causing deterioration or incapacity; or 2) investment in a system that is not optimal (that can be mass produced and that is probably less expensive), but can be readily modified (and perhaps improved) when conditions change.

Furthermore, in an environment where numerous, diverse specialized systems are derived from the general, there is an advantage of relative ease in understanding each since all have been based on the same general principles. This is contrasted with situations wherein one must learn several entirely distinct systems which are not derived from a common root. A further asset acquired from the study of a general system is its potential for engendering insights that allow what is known in one system to be translated to others.

In an earlier section ( 1.4 ) the notions of machine software and "human software" were compared. Interestingly, this comparison can be continued with respect to the question of generalization versus specialization. So-called idiot savants [ 9 ] studied by psychologists are persons capable of performing particular mental feats far beyond the capacity of normal humans, but they are unable to carry on simple conversations or perform ordinary jobs. The unusual mental feats performed by these persons, whose IQs fall in the range of 40 to 80, involve memory and prodigious calculations that surpass the abilities of the most brilliant people [15]. The savant's "software" appears to be extremely specialized, capable of such extreme concentration on a particular problem domain, that the mental "software" is not available for the wide range of problem domains addressed by more normal persons [15]. It is left to the reader to elaborate upon the interesting parallel between human and machine software with regard to the specialization versus generalization issue.

#### 4.0 Conclusion

The paper began with a generic description of decision support systems that was based upon the frameworks, surveys, and classification schemes of earlier papers [ 2 ], [ 5 ]. The generic description holds that a DSS has knowledge about a problem domain (a KS) and can accept stated problems (a LS). In order to solve a stated problem using problem domain knowledge, there must be a PPS which possesses some of the seven abilities [ 2 ] involved in decision making.

Several systems were described which fit the generic DSS idea, but which were not the customary kinds of systems encountered in business applications. The distinguishing features of each were noted. It is of interest to speculate about why systems such as MYCIN are not found in business settings. The ideas and features of such systems certainly look attractive, but in viewing management systems [ 5 ] there is nothing at this level.

This is, in part, due to the newness of the systems like those described in this paper. It is also due to the origins of such systems. PLANNER, STRIPS and MYCIN were developed by workers in the artificial intelligence field which until recently has dealt mainly with toy problems. Furthermore, workers in the field typically do not have business backgrounds. On the other hand, designers of systems such as those reviewed in [ 5 ] come predominately from a data base management-management information systems (DBM-MIS) background. The potential application of artificial intelligence techniques in conjunction with DBM-MIS techniques has yet to be widely recognized or investigated in the DBM-MIS field.

This presentation is an initial effort at remedying this situation. Research efforts directed towards the design of decision support systems that are built upon features and techniques from both traditions deserve support. Moreover, an investigation of some degree of generality that goes beyond such systems as GPLAN, is important both as a pedagogical aid and from a practical standpoint. For implementors or buyers of decision support systems, the question of precisely what level of generality is appropriate is neither a trivial problem nor a problem which has been solved. Nevertheless an understanding of a quite general system should alert these persons to the various levels of generality which are possible and it should furnish a repertoire of features and techniques which can be used to attain the desired degree of generality.

### References

- [1] R. H. Bonczek, C. W. Holsapple and A. B. Whinston, "Aiding Decision Makers with a Generalized Data Base Management System", Decision Sciences, April, 1978.
- [2] R. H. Bonczek, C. W. Holsapple and A. B. Whinston, "Computer Based Support of Organizational Decision Making", Decision Sciences, April, 1979.
- [3] R. H. Bonczek, C. W. Holsapple and A. B. Whinston, "The Integration of Data Base Management and Problem Resolution", Information Systems, Vol. 4, No. 2, 1979.
- [4] R. H. Bonczek, C. W. Holsapple and A. B. Whinston, "Representing Modeling Knowledge with First Order Predicate Calculus", submitted for publication, 1979.
- [5] R. H. Bonczek, C. W. Holsapple and A. B. Whinston, "The Evolving Roles of Models in Decision Support Systems", Decision Sciences, April, 1980. (forthcoming).
- [6] R. Davis, B. Buchanan, E. Shortliffe, "Production Rules as a Representation for a Knowledge-Based Consultation Program", Artificial Intelligence, Vol. 8, No. 1, 1977.
- [7] G. W. Ernst and A. Newell, GPS: A Case Study in Generality and Problem Solving, Academic Press, New York, 1969.
- [8] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", Artificial Intelligence, Vol. 2, 1971.
- [9] R. M. Goldenson, Encyclopedia of Human Behavior, Vol. 1, Doubleday, Garden City, New York, 1970, pp. 592-593.
- [10] M. Hammer, et. al., "A Very High Level Programming Language for Data Processing Applications", Communication of the ACM, Vol. 20, No. 11, 1977.
- [11] W. D. Haseman and A. B. Whinston, Introduction to Data Management, Irwin, Homewood, Illinois, 1977.
- [12] C. W. Holsapple and A. B. Whinston, "A Decision Support System for Area-wide Water Quality Planning", Socio-Economic Planning Sciences, Vol. 10, 1976.
- [13] A. Newell and H. A. Simon, Human Problem Solving, Prentice Hall, Englewood Cliffs, New Jersey, 1972.
- [14] E. Post, "Formal Reductions of the General Combinatorial Problem", American Journal of Mathematics, Vol. 65, 1943.

- [15] B. Rimland, "The Autistic Savant", Psychology Today, Vol. 12, No. 3, 1978.
- [16] R.A. Seaberg and C. Seaberg, "Computer Based Decision Systems in Xerox Corporate Planning, "Management Science, vol 20, no. 4, 1973.
- [17] H.A. Simon, "The Heuristic Compiler", in Representation and Meaning, (ed., H.A. Simon and L. Siklossy), Prentice-Hall, Englewood Cliffs, New Jersey, 1972.
- [18] H.A. Simon, The New Science of Management Decision, Harper & Brothers, New York, 1960.
- [19] M.H. van Emden, "Programming with Resolution Logic", in Machine Intelligence, Vol. 8, (ed., E.W. Elcock and D. Michie), Halsted Press, New York, 1977.
- [20] T. Winograd, Understanding Natural Language, Academic Press, New York, 1972, pp. 108-126.
- [21] H.K.T. Wong and J. Mylopoulos, "Two Views of Data Semantics", University of Toronto, December, 1976.